計算機アーキテクチャ　103－4
設　計　自　動　化　69－4
(1993. 12. 16)

Partial Collapsing を用いた遅延最小化の一手法

佐藤 光一　河原林 政道　江村 秀之　前田 直孝

NEC ULSI システム開発研究所

〒211 川崎市中原区下沼部 1753

あらまし

遅延最小化法には、テクノロジ依存レベルのトリーマッピング、バッファリングによる手法と、テクノロジ独立レベルで論理段数を削減するパーシャルコラップシングによる手法がある。本稿では従来のパーシャルコラップシングをテクノロジ依存レベルへも適用可能なように発展させた手法を提案する。本手法はすべてのクラスタをコラップシングせず、最大クリティカルパス上のクラスタのみコラップシングすることで効率良く遅延を最小化できた。論理合成システム Varchsyn 上に搭載したところ、実設計回路に対して従来の最適化手法のみを適用するのに比較して最大36%、平均 6% の遅延が改善した。また、テクノロジ独立レベルより、テクノロジ依存レベルに適用した方が遅延最小化性能が良く、両レベルへ適用した場合更に結果が良くなることが確認できた。

和文キーワード　CAD, 論理合成, 遅延最小化, 再合成, パーシャルコラプシング

A Delay Minimization technique using Partial Collapsing

Koichi SATO　Masamichi KAWARABAYASHI　Hideyuki EMURA　Naotaka MAEDA

NEC ULSI System Development Laboratories

1753 Shimonumabe, Nakahara-ku, Kawasaki, Kanagawa, 211

Abstract

We propose a new delay optimization technique using partial collapsing, that is extended from a previous one which aims at reducing logic level in the technology independent circuits. Our approach collapses only nodes in the cluster on the maximum critical path. It's implemented on Varchsyn (logic synthesis system), and the experiment shows the maximum delays of the circuits synthesized with this technique are improved on average 6%(max 36%) better than the conventional timing optimization sequence. In comparison with applying it to technology independent circuits, the maximum delays can be reduced efficiently when applying it to technology dependent circuits. Furthermore, we achieve a much better result when applying it both before and after technology mapping.

英文 key words　CAD, logic synthesis, delay minimization, resynthesis, partial collapsing

1 はじめに

LSIの大規模化に伴い、論理設計の自動化が進んでいる。一般にRTレベルで記述された仕様を入力し、最適化した後にネットリストを出力することを「論理合成」と呼ぶ。論理合成における回路の最小化手法の目的としては、面積最小化、遅延最小化、消費電力最小化などがある。その中で、従来、遅延最小化のアルゴリズムとして、遅延値を評価関数としたトリーマッピング、ハイパワーゲートへの置換えや負荷分散を行なうバッファリング (buffering) [7] によるテクノロジ依存レベルでの最小化が主な研究課題であった。近年、遅延が小さくなりやすいようにテクノロジ独立のレベルであらかじめ論理構造を変えておく再合成 (resynthesis) のアルゴリズムなどの研究も盛んである。

再合成のもっとも確実な方法は、回路全体を2段化して論理段数を減らし、そこでテクノロジマッピング (technology mapping) を行う方法である。この圧縮処理をコラップシング (collapsing) という。しかし、単に回路全体を2段化すると面積が膨大に増加すること、不必要な部分まで論理段数を減らしてしまうこと、多くの処理時間を必要とするなどの問題点があり現実的ではない。そのため、クリティカルパスのみの段数を小さくするように多段化する方法 [1] や、回路を部分的に2段化する方法などが研究されてきた。部分的に2段化する方法として、パーシャルコラップシング (partial collapsing) [3] がある。パーシャルコラップシングでは、クラスタと呼ばれる部分回路を求め、クラスタ中の回路を2段論理に圧縮することにより論理段数を削減し、遅延を最小化する。

しかし、従来法は最大クリティカルパス以外のクラスタにもコラップシングをかけてしまうため、コラップシング後の面積が大きくなる欠点があった。また、回路の段数で遅延を予測するため、回路全体を2入力NANDゲートなどに分解してからでないとパーシャルコラップシングができなかった。本稿ではこれらの問題を考慮した手法を提案する。本手法はテクノロジ独立、テクノロジ依存のいずれのレベルからもパーシャルコラップシング可能であり、最大クリティカルパス上のクラスタのみをコラップシングする。テクノロジ独立の場合、最大クリティカルパスは、仮想的な遅延計算式を用いて求める。テクノロジ依存の場合、ライブラリのデータを用いて正確に遅延計算する。そのため、2入力NANDに分解せずに効率よくかつ正確に遅延最小化可能である。

2節では用語の定義を行ない、3節では従来手法のパーシャルコラップシングを紹介し、4節では本手法のパーシャルコラップシングの提案を行う。5節では実験結果・考察を示す。6節では、本稿のまとめをする。

2 用語の定義

本稿で使用する用語を以下のように定義する。

リテラル数： 論理式に表れる変数の数

キューブ数： 積項の数

SOP形： 積和形。sum-of-productsの略。

factor形： $a*b+a*c$のような論理式は$a*(b+c)$で表現できる。このような形をfactor形といい、この場合のリテラル数は3となる。

ノード： 一般に論理合成では回路をグラフに置き換えて問題を解く。論理を表すゲートをノードとし、ゲート間の接続関係を有向枝で表す。

スラック： 要求時刻から到着時刻を引いた値。

クリティカルパス： スラックが負となる遅延制約を違反した信号のパス。最大クリティカルパスは出力端子から入力端子へ最小スラックをたどってできたパス。

ファンイン (fanin)、ファンアウト (fanout)： あるノードvに入力するノードをvのファンイン$FI(v)$、出力した先のノードをファンアウト$FO(v)$という。

TFI/TFO： あるノードから入力 / 出力方向へパスをたどっていくときに通るノード群。transitive fanin / fanoutの略。

3 従来手法

パーシャルコラップシング [3] は、テクノロジ独立のレベルで回路を部分的に2段化し、回路の段数を少なくすることで遅延最小化を行なう。

この手法では、Lawlerのアルゴリズム [2] を用いて、クラスタと呼ばれるパーシャルコラップシングするノード群を選択していく。アルゴリズムは以下の通りである。

Step.1 2入力NANDに分解 (図1(a))

予め入力回路を同じ大きさのノードに分解しておく。

Step.2 ラベリング (図1(b))

入力から出力へトポロジカルな順*にノードを探索し、各々のノードvに対して、そのファンインの中で最も大きなラベルL(段数) を求める。もし、vがファンインを持たない場合、vのラベルを0とする。Lと同じラベルをもつvのTFIの数kを求め、kがクラスタサイズK†を超える場合vのラベルは$L+1$、超えない場合はLとする。

Step.3 クラスタリング (図1(b))

出力から入力へ逆トポロジカルな順にノードを探索していく。ノードvのラベルがそのファンアウ

*出力端子から入力端子方向へ深さ優先探索して並んだノードの順
†クラスタ内のノードの最大数の閾値

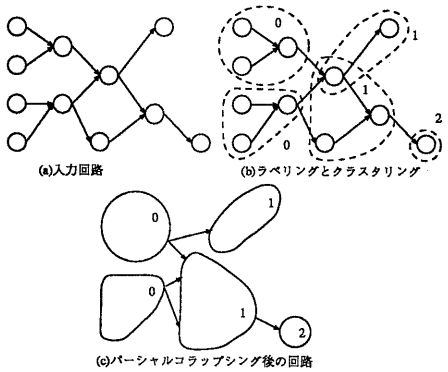

図 1: 従来手法のパーシャルコラップシング

トの全てのノードのラベルより小さいなら、v のTFIで v と同じラベルのノードとノード v で新しいクラスタを作成する。図 1(b) は図 1(a) をクラスタサイズ 3 でラベリングし、クラスタリングした時の例である。点線で囲んだものがクラスタ、クラスタについた数字がラベルである。

Step.4 コラップシング (図 1(c))

クラスタ中のノードのコラップシングを行う。二つ以上のクラスタにまたがるノードは重複して、コラップシングする。

上記のようにノードの個数でクラスタを作成するため、予め回路を同じ大きさのノードに分解しておく必要があった。ここでは、クラスタサイズ K という固定した値を与えた場合の方法について述べたが、重複率[‡]を用いて、クラスタサイズを自動計算する方法がある。また、面積を増やさないようにするため、すべての出力端子のラベルを最大値に置き換えてラベリングし直す方法もある。

4 パーシャルコラップシングの新手法

4.1 従来手法の問題点と新手法

パーシャルコラップシングにより面積増加をおさえて、遅延最小化を行うことができるようになった。しかし、従来手法には以下のような問題点があった。

1. コラップシング後の面積が大きくなる。
2. 回路の段数で遅延を予測するため、回路全体を 2 入力 NAND ゲートに分解する必要がある。

1 つ目の問題点の原因は、テクノロジ独立のレベルでの遅延を精度良く見積もる手段が無く、クリティカルパスを見つけることができず、全体をコラップシングしなければならなかったためである。また、テクノロジ依存レベルでのパーシャルコラップシングを考慮していなかったことも理由の 1 つと考えられる。

2 つ目の問題点は、クラスタサイズの指標をノード個数に設定していたことである。2 入力 NAND などの一定の大きさのノードに分解する必要があり、テクノロジ依存レベルで効果的にパーシャルコラップシングを適用できない。

このような問題点を解決するパーシャルコラップシングの手法を提案する。本手法は以下のような特徴を持つ。

- 最大クリティカルパス上のクラスタのみコラップシングする。テクノロジ独立の場合、最大クリティカルパスは、テクノロジ独立用の遅延計算式を用いて求める。テクノロジ依存レベルの場合、テクノロジライブラリブロックのデータを用いて正確に遅延計算する。
- 2 入力 NAND に分解せず、入力回路のノードを直接コラップシングする。本手法はリテラル数をノードの重みとして加算し、クラスタサイズを算出する。

1 つ目の特徴は問題点 1 を解決するため、2 つ目は問題点 2 を解決するためのものである。以下では、本手法の実現にあたり必要となるテクノロジ独立の遅延計算法、アルゴリズムについて示す。

4.2 テクノロジ独立の遅延計算法

本手法は最大クリティカルパスを算出し、その上のクラスタのノードのみコラップシングする。テクノロジ依存レベルの場合、ライブラリからそのゲートの内部遅延、配線遅延を用いて正確に遅延計算できる。テクノロジ独立なレベルでの遅延見積方法 [4],[5] はいくつか研究されており、本手法では [4] の手法を拡張している。

$$d = a_0 + a_1 \ln c + a_2 \ln f \tag{1}$$

$$c = t \cdot s \tag{2}$$

d はノードの遅延、t はキューブ数、s はキューブのリテラル数の最大値、c は論理関数の複雑度、f はノードのファンアウト数を表す。a_0, a_1, a_2 はテクノロジ依存のパラメータである。ノードは SOP 形で考え、式 (2) で論理関数の複雑度を計算する。

テクノロジ依存の遅延計算法と同様に、式 (1) を論理関数の複雑度に依存する式 (3) とファンアウトに依存する式 (4) に分解して考えることができる。

$$d_1 = b_0 + b_1 \ln c \tag{3}$$

[‡]コラップシングすることで増加するノード数とコラップシング前の全ノード数の比率

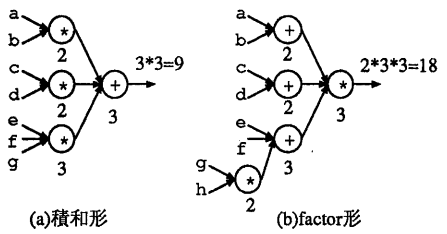

図 2: テクノロジ独立遅延計算における論理関数複雑度

$$d_2 = c_0 + c_1 \ln f \tag{4}$$

d_1 は論理関数の複雑度 c に依存する遅延、d_2 はファンアウト数 f に依存する遅延でこれらの和がノードの遅延 d となる。b_0, b_1, c_0, c_1 はテクノロジに依存するパラメータである。

[例 1]
$a*b+c*d+e*f*g$ のような式の場合、最大のリテラル数をもつキューブのリテラル数は 3、項数は 3 なので $3*3=9$ が論理複雑度となる (図 2(a))。

本手法では、SOP 形で計算する方法のほかに、factor 形で論理関数の複雑度を計算する方法を提案し、導入している。ノードを factor 形にし、遅延計算には式 (1) を用い、複雑度は式 (5) で計算する。

$$c_{i+1}^k = \max_{1 \le j \le t_i^k} (c_i^j) \cdot t_i^k \tag{5}$$

$$c_2^k = t_1^k \tag{6}$$

c_{i+1}^k は $i+1$ 回目の論理演算における k 番目の項の論理複雑度を表す。t_i^k は $i+1$ 回目の論理演算において k 番目の項となる部分の i 回目の論理演算での項数である。

[例 2](図 2(b))
$(a+b)*(c+d)*(e+f+(g*h))$ のような式の場合、まず、$g*h$ で 2 を得る。これは図 2(b) の g,h の入力が入るゲートに相当する。次に $e+f+(g*h)$ で e は論理複雑度が 1、f は論理複雑度が 1、$g*h$ は論理複雑度が 2 なので、2 が次の論理演算における最大論理複雑度となる。$e+f+(g*h)$ は $2*3=6$ となる。同様にして、$(a+b)*(c+d)*(e+f+(g*h))$ の論理複雑度は、$6*3=18$ となる。

4.3 アルゴリズム

4.3.1 3 手法

以下の 3 種類のパーシャルコラップシングの制約指定手法を試みた。

1. SIZE

   クラスタサイズを直接指定する。指定クラスタサイズでパーシャルコラップシングを行なう。

2. BEST

   全体重複率§を指定する。

   まず、全体重複率以内に収まる最大クラスタサイズを計算する。このクラスタサイズでパーシャルコラップシングを行なう。

3. CRIT

   全体重複率と部分重複率¶を指定する。

   最小スラック出力端子の TFI で構成した部分回路の最大クラスタサイズを部分重複率で計算し、その部分回路のパーシャルコラップシングを行なう。

   この操作を要求時刻を満たすまで繰り返す。

   但し、部分重複率で計算した重複数が全体重複数‖を超える場合、部分重複率を全体重複数を超えない最大値に変えて最大クラスタサイズを求め、パーシャルコラップシングを行ない、処理を終了する。

いずれの手法も途中ですべての出力が要求時刻を満たすと分かった場合、終了する。

4.3.2 SIZE

クラスタサイズ指定の場合のアルゴリズムを示す。

まず、あらかじめマッピングするライブラリに応じた遅延モデルのパラメータ a_0, a_1, a_2 を求めておく。ノードにはリテラル数を重みとして与え、遅延計算にはテクノロジ 独立の場合は式 (1) を用い、テクノロジ依存の場合はライブラリのデータを用いた正確な遅延計算を用いる。

パーシャルコラップシングは以下の手順で行なう。

Step.1 *lev*, *alev* の設定

従来法はノードの重みをノードの個数としていたが、本手法では重みをリテラル数で考えるので 1 つのノードでクラスタサイズ K を超える場合がありうる。そのため、step.1で 1 つのノードで占

§コラップシングの際増加するノードのリテラル数とコラップシング前の回路の全リテラル数の比率

¶部分回路コラップシングの際増加するノードのリテラル数とコラップシング前の部分回路の全リテラル数の比率

‖全体重複率から計算した重複数

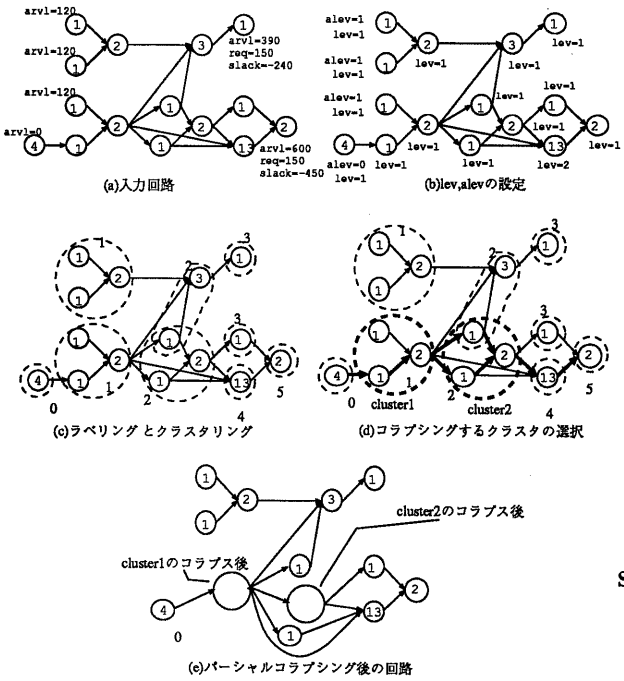

図 3: 本手法のパーシャルコラップシング

めるクラスタの数 (lev と名づける) を計算する必要がある。

ノードの重みがクラスタサイズ K 以内なら、$lev = 1$ とする。ノードの重みがクラスタサイズ K 以上の場合には以下のようにして計算する

回路がマッピングされていない場合、

$$lev = \frac{a_0 + a_1 \ln c}{a_0 + a_1 \ln K} \qquad (7)$$

でノード v の $lev(v)$**を計算する。c は論理複雑度である。

回路がマッピングされている場合、

$$lev = \frac{delay}{a_0 + a_1 \ln K} \qquad (8)$$

でノード v の $lev(v)$ を計算する。$delay$ はノード v のそれぞれの入力から出力までのパス中の最大遅延時間であり、a_0, a_1 は式 (1) の遅延パラメータ、K はクラスタサイズである。

また、従来法は全ての入力の到着時刻が全て等しいことを想定して入力端子のラベルを 0 から始めていた。本手法は入力端子の到着時刻が様々なことを考慮して、ラベル $alev$ から始まるようにする。$alev$ は以下のようにして計算する。

$$alev = \frac{arrival\ time}{a_0 + a_1 \ln K} \qquad (9)$$

$arrival\ time$ は入力端子の到着時刻、a_0, a_1 は式 (1) の遅延パラメータ、K はクラスタサイズである。ただし、入力端子以外は $alev = 0$ となる。

図 3(a) のようなテクノロジ独立の回路に、クラスタサイズ 4 でのパーシャルコラップシングをしたときを例に取り上げる。円がノードを表し、その中の数字がリテラル数である。矢印の方向は入力から出力方向への信号の方向を表す。arvl が到着時刻、req が要求時刻、slack がスラックである。このときの遅延パラメータは $a_0 = 30$, $a_1 = 60$, $a_2 = 10$ とした。$lev, alev$ は式 (7), (9) で算出し、図 3(b) のような結果が得られたとする。

Step.2 ラベリング

ラベリングは従来と同様の操作をする。ただし、スタートラベルを 0 ではなく $alev$ とし、隣合うクラスタのラベルの差は 1 ではなく lev で操作される。アルゴリズムは以下の通りである。

```
foreach(ノードv ∈ トポロジカル順に入力端子から出力端
子へ探索) {
  if(vが入力端子) then L = 0;
                  else L= max(label(n));
                          n ∈ FI(v)
  if(alev(v) > L) then L=alev(v);
  if(vの重み > クラスタサイズK)
  then label(v) = L+lev(v)
  else if(Σ TFI(v) 中の label Lのノードのリテラル
                      + vのリテラル数 ≧ K)
       then label(v) = L+1;
       else label(v) = L;
}
```

Step.3 クラスタリング

クラスタリングは従来手法と同様である。図 3(a) でラベリング、クラスタリングすると図 3(c) のようになる。点線で囲んだものがクラスタ、数字がそれぞれのクラスタ中のノードのラベルである。

Step.4 コラップシング

コラップシングでは、全てのクラスタをコラップシングせずに、最大クリティカルパスのクラスタのみコラップシングする。図 3(d) の太矢印が、最大クリティカルパスである。最大クリティカルパスのクラスタで、その中に 2 つ以上のノードをもつものが処理対象となるので、この場合 cluster1 と cluster2 が選択され、コラップシングする

**lev, $alev$ は小数点以下を四捨五入して整数化する

候補となる。コラップシングするクラスタは式(10)の評価関数を用いて決定する。

$$value(c) = \alpha \cdot \sum_{i=1}^{n(c)} slack(c)_i + \beta \cdot dup(c) \quad (10)$$

評価値は、スラック最小のノードに対して求める。複数の最大クリティカルパスがあるときその多くのパスに存在し、コラップシングによって重複することが少ないクラスタが選択される。$slack(c)_i$ はクラスタ c のスラックがその TFO である i 番目の出力端子のスラックと等しいときその出力端子のスラックになり、等しくない時 0 となる。$n(c)$ はクラスタ c の TFO となる出力端子の数である。$dup(c)$ はクラスタの重複するノードの重み総和である。テクノロジ独立なレベルの場合、複数の最大クリティカルパスが存在することがよくあるので評価値に最小スラックの和を求める式を加えているのである。α, β は遅延時間と面積のどちらに重きをおくかを表す係数である。α が大きくなると遅延時間の注目度が高くなり、β が大きくなるとその逆となる。

$\alpha = 1, \beta = 1$ のとき、クラスタ cluster1 が先に選択され、コラップシングする。回路がマッピングされている場合は、そのコラップシングされた部分のみマッピングし直す。但し、クラスタのコラップシング後、そのルートノード††の遅延値が悪化した場合、コラップシング前の状態に戻す。もし、ここで要求時刻の全てを満たすようなら処理を終えるが、そうでないならもう一度タイミング解析を行い、最大クリティカルパスを求め、クラスタ選択、コラップシングの操作を繰り返す。コラップシングできるクラスタがなくなった場合はその時点で終了する。図 3(e) に最後までコラップシングしたときの結果を示す。

4.3.3 BEST

ラベリングの前に行うクラスタサイズの計算以外は SIZE の処理と全く同じである。

クラスタサイズは 2,4,8,16,.. と増加させて指定重複率を超えるクラスタサイズ 2^n を求めた後、2^n と 2^{n-1} の間にある重複率を超えない最大クラスタサイズを求める。これは 2 分探索を用いて求める。この求めたクラスタサイズでパーシャルコラップシングを行なう。

4.3.4 CRIT

BEST では回路全体を一括して扱うが、1 つの出力端子のスラックが大きい場合にはそのクリティカルな部分の回路のみクラスタサイズを大きくした方が効果があると考えられる。

CRIT は最もクリティカルな部分回路からパーシャルコラップシングする。アルゴリズム自体は BEST とほとんど同じだが、クラスタサイズの計算は最もクリティカルと考えられる部分回路に対して行ない、求めたクラスタサイズでその部分回路に対してのみパーシャルコラップシングを行なう。

5 実験結果・考察

論理合成システム Varchsyn[6] 上で同一ライブラリ ($1\mu m$ のプロセスのゲートアレイ) を用いて実験した結果を示す。パーシャルコラップシングは、通常の遅延優先最小化手順の面積優先マッピング前／後に挿入した。要求時刻や FF のクロックの設定はパーシャルコラップシングを用いた時も用いなかった時も全く同じ条件である。入力端子における到着時刻は 3 つの手法すべて $0ns$ とした。パーシャルコラップシングの評価値計算パラメータは $\alpha = 1, \beta = 1$ である。

3 つの手法のパーシャルコラップシングを使った場合と、使わなかった場合の性能の比較を表 1, 2に示す。データには MCNC ベンチマークデータを用いた。 nopc はパーシャルコラップシングを用いなかった場合の最大遅延値、SIZE, BEST, CRIT はパーシャルコラップシングで用いた前述の手法を表す。SIZE はクラスタサイズ 8、BEST は全体重複率 0.3、CRIT は全体重複率, 部分重複率ともに 0.3 である。 indep. はノードがマッピングされる前、dep. はノードがマッピングされた後にパーシャルコラップシングを適用した結果を表す。() 中は nopc を 1 とした時の性能比である。遅延モデルについてデータを見る限りでは、テクノロジ依存で SOP 形を使用した方が良好な結果を得ることができた。この原因は、factor 形にすると EXOR のリテラル数を 2 としているので、評価が 2 入力 NAND と同じ位になり遅延値を小さく見積もってしまうため、コラップシング後遅延がかえって悪くなると判定されて元の状態に戻すことが多くなるためと考えられる。この実験を見る限りでは、3 種類のうちのどの手法が最も優れているとはいいがたいが、いずれもこれまでの遅延最小化手順よりも良い結果が得られた。

パーシャルコラップシング をテクノロジ独立 (FF はすでにマッピングされているものとする)、テクノロジ依存のレベルで行なった場合の性能比較を表 3に示す。データは、MCNC,ISCAS のデータ 18 個と、VHDL 記述の実設計データ 19 個を用いた。遅延モデルには SOP 形、手法としては BEST(全体重複率 0.3) を用いた。 nopc がパーシャルコラップシングを用いなかった場合、indep. はノードがマッピングされていない時、dep. はノードがマッピングされている時にパーシャルコラップシングを適用した結果を表す。面積の単位はセル数で 1 セルはインバータ 1 個の大きさに相当する。処理時間はデータの入力からネットリストを出力するまでの全フローの処理時間である。テクノロジ独立の時、最大で 33.1% の遅延削減が可能で、最悪 11.9% 遅延を悪くする (遅延が

††クラスタ内の圧縮対象ノード

表 1: 最大遅延比較と平均面積増加量 (SOP 形, MCNC ベンチマークデータ, 遅延単位 1/100ns)

circuit	nopc	partial collapsing					
		SIZE		BEST		CRIT	
		indep.	dep.	indep.	dep.	indep.	dep.
5xp1	816	825 (1.01)	802 (0.98)	880 (1.08)	847 (1.04)	825 (1.01)	817 (1.00)
9sym	1401	1171 (0.84)	1302 (0.93)	1192 (0.85)	1278 (0.91)	1192 (0.85)	1278 (0.91)
bw	655	681 (1.04)	665 (1.01)	680 (1.04)	684 (1.04)	723 (1.10)	665 (1.01)
duke2	1118	1036 (0.93)	947 (0.85)	1036 (0.93)	1044 (0.93)	1015 (0.91)	1003 (0.90)
f2	424	424 (1.00)	342 (0.81)	424 (1.00)	367 (0.87)	424 (1.00)	364 (0.86)
rd53	760	778 (1.02)	749 (0.99)	778 (1.02)	655 (0.86)	760 (1.00)	655 (0.86)
rd73	1024	875 (0.85)	895 (0.87)	875 (0.85)	951 (0.93)	875 (0.85)	1024 (1.00)
sao1	1095	1034 (0.94)	1061 (0.97)	1034 (0.94)	1042 (0.95)	1034 (0.94)	1095 (1.00)
sao2	914	881 (0.96)	890 (0.97)	881 (0.96)	890 (0.97)	866 (0.95)	883 (0.97)
vg2	822	822 (1.00)	746 (0.91)	711 (0.86)	776 (0.94)	774 (0.94)	806 (0.98)
遅延比平均		0.96	0.93	0.95	0.94	0.96	0.95
面積比平均		1.06	1.25	1.11	1.18	1.08	1.21

表 2: 最大遅延比較と平均面積増加量 (factor 形, MCNC ベンチマークデータ, 遅延単位 1/100ns)

circuit	nopc	partial collapsing					
		SIZE		BEST		CRIT	
		indep.	dep.	indep.	dep.	indep.	dep.
5xp1	816	825 (1.01)	877 (1.07)	880 (1.08)	752 (0.92)	825 (1.01)	791 (0.97)
9sym	1401	1378 (0.98)	1260 (0.90)	1378 (0.98)	1260 (0.90)	1378 (0.98)	1260 (0.90)
bw	655	666 (1.02)	665 (1.01)	680 (1.04)	664 (1.01)	723 (1.10)	672 (1.03)
duke2	1118	1036 (0.93)	1001 (0.90)	1036 (0.93)	1048 (0.94)	1015 (0.91)	1049 (0.94)
f2	424	424 (1.00)	341 (0.80)	424 (1.00)	367 (0.87)	424 (1.00)	364 (0.86)
rd53	760	760 (1.00)	673 (0.89)	760 (1.00)	748 (0.98)	760 (1.00)	748 (0.98)
rd73	1024	1024 (1.00)	917 (0.90)	1024 (1.00)	912 (0.89)	1024 (1.00)	896 (0.88)
sao1	1095	1049 (0.96)	1034 (0.94)	1049 (0.96)	1063 (0.97)	1049 (0.96)	1063 (0.97)
sao2	914	881 (0.96)	901 (0.99)	873 (0.95)	901 (0.99)	866 (0.95)	797 (0.87)
vg2	822	822 (1.00)	825 (1.00)	711 (0.86)	825 (1.00)	774 (0.94)	825 (1.00)
遅延比平均		0.99	0.94	0.98	0.95	0.98	0.94
面積比平均		1.01	1.27	1.01	1.23	0.99	1.19

悪くなるものは 37 個中 6 個)。テクノロジ依存の時は、最大で 36.2% の遅延削減が可能で、最悪 6.2% 遅延を悪くする (遅延が悪くなるものは 37 個中 6 個)。この結果によると特に実回路で有効であることが分かる。平均ではテクノロジ独立の時、5.2% の遅延削減、2.9% の面積増加、41.9% の処理時間増加となり、テクノロジ依存の時、6.4% の遅延削減、12.4% の面積増加、15.9% の処理時間増加となる。テクノロジ 独立よりも、テクノロジ依存で用いた方が遅延削減が可能である。この原因として、テクノロジ依存の方が正確にコラプッシングを採用するかどうかを判定可能なためと考えられる。パーシャルコラップシングを適用しなかったものより悪くなる原因は、パーシャルコラップシングによって回路構成が変わり、その後で行なう遅延改善手法がうまく働かなかったためと考えられる。また、テクノロジ依存の方が処理時間が短くなる原因は、独立の場合よりも早く収束した結果に近くなるためパーシャルコラップシング後の遅延改善手法において改善の余地が少なく処理時間が短くなることと、テクノロジ独立の方が一つのノードが大きく、クラスタサイズを大きく設定しやすいためコラップシングに時間がかかることが考えられる。

パーシャルコラップシングの評価関数を変更し、最大でないクリティカルパス上のクラスタもコラップシングして表 3 と同じ実験をした時の結果の平均を表 4 の 3 段目,4 段目に示す。最小スラック以外のものも含めて、式 (10) で計算する。但し、最小スラック以外も対象とするため、$slack(c)_i$ はクラスタ c のスラックがその TFO である i 番目の出力端子のスラックより小さいときその出力端子のスラックになり、そうでない時 0 とする。回路データは、表 3 のデータと同じである。平均でテクノロジ独立の場合、遅延削減 4.3%、面積増加 3.4%、処理時間 43.5% 増加となり、テクノロジ依存の場合、遅延削減 5.4%、面積増加 22.1%、処理時間 23.3% 増加となった。テクノロジ独立の場合も依存の場合も、最大クリティカルパスのみコラップシングしたもののほうがすべての項目において良い結果を得ることがわかる。最大遅延でも悪くなるのは、最大クリティカルパス以外のノードのコラップシングによりノードが重複し、最大クリティカルパス上のノードのファンアウト数が増え遅延が増加する場合があるからと考えられる。

パーシャルコラップシングをテクノロジ独立 (FF はすでにマッピングされているものとする) とテクノロジ依存の両レベルで適用した時の結果の平均を表 4 の 5 段目に示す。遅延モデルには SOP 形、手法としては BEST(全体重複率 0.3) を用いた。回路データは、表 3 のデータと同じである。7.5% の遅延削減、14.0% の面積増加、11.5% の処理時間増加である。表 4 の 2 段目のテクノロジ依存の場合よりもわずかに 1.6% 程面積増加するが、テクノロジ独立、依存のみの場合よりさらに遅延削減が可能で、処理時間も最も短い。処理時間が短いのは、パーシャルコラップシングを用いた方が他手法での改善の余地が少なく早期に収束した結果に近づくためと考えられる。

表 3: テクノロジ独立 .vs. テクノロジ依存 の性能比較

circuit	最大遅延 1/100ns			面積 セル			処理時間 sec.		
	nopc	indep.	dep.	nopc	indep.	dep.	nopc	indep.	dep.
5xp1	816	880 (1.078)	847 (1.038)	208	167 (0.803)	230 (1.106)	205.3	117.8 (0.574)	82.7 (0.403)
9sym	1401	1192 (0.851)	1278 (0.912)	70	101 (1.443)	93 (1.329)	21.1	35.9 (1.701)	34.0 (1.611)
bw	655	680 (1.038)	684 (1.044)	328	276 (0.841)	272 (0.829)	324.7	175.0 (0.539)	208.8 (0.643)
duke2	1118	1036 (0.927)	1044 (0.934)	505	547 (1.083)	623 (1.234)	396.7	458.0 (1.155)	369.7 (0.932)
f2	424	424 (1.000)	367 (0.866)	33	48 (1.455)	49 (1.485)	8.9	12.4 (1.393)	9.3 (1.045)
rd53	760	778 (1.024)	655 (0.862)	34	33 (0.971)	57 (1.676)	8.7	13.3 (1.529)	20.0 (2.299)
rd73	1024	875 (0.854)	951 (0.929)	50	63 (1.260)	55 (1.100)	21.6	12.8 (0.593)	20.3 (0.940)
sao1	1095	1034 (0.944)	1042 (0.952)	80	87 (1.087)	83 (1.038)	32.3	33.9 (1.050)	35.3 (1.093)
sao2	914	881 (0.964)	890 (0.974)	267	290 (1.086)	291 (1.090)	216.0	332.7 (1.540)	172.2 (0.797)
vg2	822	711 (0.865)	776 (0.944)	191	203 (1.063)	165 (0.864)	118.7	178.8 (1.506)	110.5 (0.931)
C1355	2784	2118 (0.761)	2292 (0.823)	895	608 (0.679)	1237 (1.382)	2033.2	1111.0 (0.546)	2895.3 (1.424)
C1908	3009	2926 (0.972)	2939 (0.977)	619	684 (1.105)	670 (1.082)	843.6	885.4 (1.050)	906.7 (1.075)
C2670	2704	2379 (0.880)	2408 (0.891)	947	1131 (1.194)	1191 (1.258)	1560.1	1812.1 (1.162)	2311.7 (1.482)
C432	2546	2623 (1.030)	2675 (1.051)	508	350 (0.689)	507 (0.998)	377.6	299.2 (0.792)	414.4 (1.097)
C499	2155	2105 (0.977)	2096 (0.973)	580	584 (1.007)	646 (1.114)	887.7	914.5 (1.030)	1403.3 (1.581)
C880	1930	1842 (0.954)	1756 (0.910)	484	475 (0.981)	618 (1.277)	317.2	282.8 (0.892)	457.5 (1.442)
C3540	3781	3499 (0.925)	3615 (0.956)	1843	1796 (0.974)	1888 (1.024)	2620.5	3445.5 (1.315)	3201.7 (1.222)
C5315	3481	3424 (0.984)	3401 (0.977)	2004	2150 (1.073)	2159 (1.077)	4723.9	6608.3 (1.399)	6059.6 (1.283)
VHDL1	1101	1081 (0.982)	1041 (0.946)	1255	1085 (0.865)	1673 (1.333)	2849.0	1031.5 (0.362)	1320.9 (0.464)
VHDL2	2217	2027 (0.914)	2189 (0.987)	1871	2303 (1.231)	2166 (1.158)	867.3	2011.7 (2.319)	981.5 (1.132)
VHDL3	1753	1724 (0.983)	1759 (1.003)	2435	2468 (1.014)	2379 (0.977)	1357.8	1201.6 (0.885)	1690.6 (1.245)
VHDL4	2100	1908 (0.909)	2035 (0.969)	1877	2460 (1.311)	1922 (1.024)	524.6	1209.6 (2.306)	1166.9 (2.224)
VHDL5	1797	1778 (0.989)	1776 (0.988)	2237	2366 (1.058)	2340 (1.046)	1309.7	1391.3 (1.062)	1206.6 (0.921)
VHDL6	1297	1220 (0.941)	1176 (0.907)	314	332 (1.057)	385 (1.226)	157.6	207.9 (1.319)	103.8 (0.659)
VHDL7	1064	1061 (0.997)	1130 (1.062)	1383	1046 (0.756)	1112 (0.804)	2763.6	933.5 (0.338)	2058.3 (0.745)
VHDL8	2029	1898 (0.935)	1710 (0.843)	222	218 (0.982)	260 (1.171)	157.9	107.5 (0.681)	210.0 (1.330)
VHDL9	1914	1914 (1.000)	1914 (1.000)	220	220 (1.000)	220 (1.000)	62.7	60.3 (0.962)	59.9 (0.955)
VHDL10	2426	2426 (1.000)	2426 (1.000)	663	663 (1.000)	663 (1.000)	79.3	78.2 (0.986)	82.1 (1.035)
VHDL11	2650	2650 (1.000)	2650 (1.000)	451	451 (1.000)	451 (1.000)	727.9	726.2 (0.998)	723.0 (0.993)
VHDL12	863	935 (1.083)	860 (0.997)	669	611 (0.913)	451 (0.674)	398.2	278.8 (0.700)	400.2 (1.005)
VHDL13	695	778 (1.119)	695 (1.000)	194	203 (1.046)	194 (1.000)	118.2	149.6 (1.266)	121.0 (1.024)
VHDL14	1728	1653 (0.957)	1621 (0.938)	975	847 (0.869)	966 (0.991)	620.7	564.1 (0.909)	767.5 (1.237)
VHDL15	1236	1236 (1.000)	1246 (1.008)	368	368 (1.000)	393 (1.068)	87.0	85.5 (0.983)	132.1 (1.518)
VHDL16	2260	1898 (0.840)	1687 (0.746)	352	372 (1.057)	442 (1.256)	203.9	458.5 (2.249)	133.7 (0.656)
VHDL17	1664	1318 (0.792)	1281 (0.770)	742	862 (1.162)	879 (1.185)	117.9	727.7 (6.172)	298.9 (2.535)
VHDL18	2002	1867 (0.933)	1658 (0.828)	1152	1038 (0.901)	1350 (1.172)	1231.6	902.4 (0.733)	1453.8 (1.180)
VHDL19	2714	1817 (0.669)	1732 (0.638)	494	518 (1.049)	759 (1.536)	252.3	1889.9 (7.491)	181.4 (0.719)
平均		0.948	0.936		1.029	1.124		1.419	1.159

表 4: 各手法の平均性能比較

手法	最大遅延比	面積比	処理時間比
テクノロジ独立 (最大クリティカルパス)	0.948	1.029	1.419
テクノロジ依存 (最大クリティカルパス)	0.936	1.124	1.159
テクノロジ独立 (クリティカルパス)	0.957	1.034	1.435
テクノロジ依存 (クリティカルパス)	0.946	1.221	1.233
テクノロジ独立と依存 (最大クリティカルパス)	0.925	1.140	1.115

6 おわりに

本稿では、テクノロジ独立、依存のどちらのレベルでも適用可能なパーシャルコラップシングを提案した。本手法は、リテラル数でクラスタを作成し、タイミング解析で最大クリティカルパスを探し、その上にのるクラスタのみコラップシングすることで最大遅延を小さくする。実験から、最大クリティカルパス上のクラスタのみコラップシングした方が最大遅延を小さくするのに有効であることが分かった。従来はテクノロジ独立でのみ考えられていたがテクノロジ依存の方がより多くの遅延削減が可能であり、テクノロジ独立と依存の両レベルで用いると面積のわずかな増加で遅延削減、処理時間短縮が可能である。特に実回路においてその効果は大きく、実用に向いた手法といえる。

今後の課題としては、クラスタ化する部分回路の選択方法、パーシャルコラップシングと他手法との組合せなどがあげられる。

参考文献

[1] K.J.Singh, A.R.Wang, R.K.Brayton, and A. Sangiovanni -Vincentelli,"Timing Optimization of Combinational Logic," ICCAD-88, pp.282-285, November 1988

[2] E.L.Lawler, K.L.Levitt, and J.Turner, "Module clustering to minimize delay in digital networks," IEEE Trans. on Comp., C-18(1):47-57, January 1969

[3] H.J.Touati, H.Savoj, and R.K.Brayton , "Delay Optimization of Combinational Logic Circuits by Clustering and Partial Collapsing," ICCAD-91, pp.188-191, November 1991

[4] D.E.Wallace, M.S.Chandrasekhar,"High-Level Delay Estimation for Technology-Independent Logic Equations," ICCAD-1990, pp.188-191, November 1990

[5] P.T.Gutwin, P.C.McGeer, "Delay Prediction for Technology-Independent Logic Equations," ACM / SIGDA TAU'92, 1992

[6] 前田 他, "Varchsyn(1): 論理合成システムの全体構成," 第 46 回情処全大 (6), pp.155-156, 1993

[7] 浅香 他, "Varchsyn(8): タイミング最適化 II," 第 46 回情処全大 (6), pp.169-170, 1993